2.2 MOVEMENT

Every player with a mover system in Suppressor is capable of moving. The location is the component of a player which moves. Each mover system has capabilities which determine the attributes (e.g. speed, acceleration, altitude, climb/dive rates, and fuel-usage) of the movement.

Players which move in Suppressor usually start with a pre-planned path defined in the SDB input file. Players may be restricted to only follow their pre-planned path, or they may dynamically change their paths based on conditions during the simulation. Dynamic changes in a player's path are defined in a MOVE-PLANS data item which is part of the tactics belonging to a player. Options for dynamic changes in movement include: suspending and resuming movement, stopping or resuming terrain following, following a repeating pattern (orbit), following a weapon delivery profile, and several more.

Suppressor also includes features for dynamically launching aircraft from an alert state or creating new players (e.g. firing missile players) during a simulation. These are both forms of moving players in Suppressor which do not start with a pre-planned path. All of the dynamic reactive movement features in MOVE-PLANS can also be used with these players.

2.2.1 Functional Element Design Requirements

The design requirements necessary to implement the movement functional element are:

a. Provide a capability for a user to define movement constraints and characteristics for each type of platform. The user options will include the following input data items.

   1. Fuel usage defined as a burn rate dependent on altitude and speed.
   2. Maximum acceleration.
   3. Maximum deceleration.
   4. Maximum traversable slope for a surface mover. (Not currently implemented)
   5. Minimum turn radius defined as a 3-dimensional distance.
   6. Maximum and minimum altitudes (MSL).
   7. Climb/dive limits defined as maximum and minimum vertical rates.
   8. Orientation limits defined as the maximum roll, pitch, and yaw angles relative to the velocity vector. (Not currently used)
   9. Orientation rates defined as maximum angular rates of change in each of roll, pitch, and yaw. (Not currently used)
   10. Maximum and minimum speed limits.
   11. Delay time between the decision to start moving and the actual start of physical movement

b. Provide a capability for the user to define a planned movement path for each player. The path will include a time to start movement and sequence of locations. The locations may be defined in terms of Cartesian coordinates or latitude, longitude, and altitude. Path altitudes (or z-coordinates) must be defined as AGL or MSL. The path must be defined as surface or three-dimensional movement, and the turns must be specified as instantaneous with sharp corners or as requiring a minimum turn radius. The following additional options will be available; all except the first may be specified for each path point.

   1. Minimum altitude AGL or maximum altitude MSL within a (2-dimensional) polygonal area
   2. Climb/dive rate or climb/dive angle
   3. Terrain-following, terrain-avoidance, or threat-avoidance
   4. Turn mode as an instantaneous turn (sharp corner) or as a turn in an arc that starts at the given path point
   5. Speed and speed changes
   6. Stop at this point for a specified time interval
   7. Turn direction as right or left or the shorter of the two directions.
   8. Turn radius in terms of distance or force (g's) or roll, pitch, and yaw (Only the distance option is currently used)

c. Provide a capability for the user to define reactive movement paths for each type of platform in the TDB. The path will include a sequence of locations defined as relative Cartesian coordinates. The x and y coordinates may be relative to the mover or relative to the target; the z-coordinate may be defined as AGL, MSL, relative to the mover, or relative to the target. The following additional options for each path point will be available.

   1. Climb/dive rate or climb/dive angle
   2. Terrain-following or terrain-avoidance or threat-avoidance
   3. Turn mode as an instantaneous turn (sharp corner) or as a turn in an arc that starts at the given path point
   4. Speed and speed changes.
   5. Stop at this point for a specified time interval
   6. Turn direction as right or left or the shorter of the two directions.
   7. Turn radius in terms of distance or force (g's) or roll, pitch, and yaw (Only the distance option is currently used)

2.2.2 Functional Element Design Approach

The basic algorithm for movement paths in Suppressor is to define a desired path with a series of points (x,y,z and speed). The desired series of points is defined from the input path points, the projected target intercept position, the points in a pattern, the points in a weapon delivery profile, or the points necessary to perform terrain following. The

algorithm then creates a path data structure by connecting the points using straight lines or circular arcs in 3-dimensions.

Suppressor may limit the actual attainable speed or altitude. The movement algorithm accounts for this by comparing the desired speed and altitude with user defined movement capabilities to ensure the resultant path is within the movement limits of the platform. The primary TDB data items which represent the propulsion limits for a player are MAX-ACCELERATION and MOVER-SPEED-LIMITS. The data item, MAX-ACCELERATION, is a single value that limits the rate at which a mover can change its speed. The MOVER-SPEED-LIMITS data item includes two values which define the lowest and highest speed a mover is capable of using.

Reactive movement is implemented through one or more MOVE-PLANS data items in the TDB. Available tactical criteria allow the transitioning between different MOVE-PLANS and the execution of user-defined profiles and patterns defined in the PLAN-PROFILE and PLAN-PATTERN data items. Execution of a MOVE-PLANS, is achieved by reference to it in the SDB PATH or PLANS-FOR-MOVEMENT data items.

2.2.3 Functional Element Software Design

Suppressor uses four top-level software modules, listed in Table 2.2-1, to implement player movement.

TABLE 2.2-1. Movement-Related Software Modules

Modules	Data Block (number)	Data Items
DYNPTH DYNVEC DYNTRN MAXACC	Movement Related Info. (42) Movement Limits (43)	MAX-ACCELERATION MOVER-SPEED-LIMITS

Subroutine DYNPTH is the primary module used for adding points to a movement path and ensuring the data are within the propulsion and other limits of the mover. This is the top-level design for Subroutine DYNPTH:

```
ABSTRACT     add new point to movement path
   *begin logic to add new point to movement path:
      *compute vector straight between points;
      *when new point is different from previous point:
         *copy values that may change due to limits;
         *when mover-altitude-limits apply and are exceeded:
            *adjust path altitude to within altitude-limits;
            *write error message informing of changed altitude;
         *end of test if mover-altitude-limits apply and exceeded.
         *when mover-speed-limits apply and are exceeded:
            *adjust path speed to within speed-limits;
            *write error message informing of changed speed;
         *end of test if mover-speed-limits apply and are exceeded:
         *allocate new FPL point;
         *add new point to end of future path list;
         *store position and speed;
         *invoke logic to normalize linear vector between points;
         *when no arc path or no speed at previous point:
            *store unit velocity vector in previous and new point;
            *when constant (default) acceleration mode:
```

```
            *but, when maximum acceleration mode:
            *end of test for acceleration mode.
         *otherwise, arc is allowed:
            *invoke logic to define plane of maneuver vectors;
            *when it is a linear path:
               *store unit velocity vector in previous and new point;
               *when constant (default) acceleration mode:
               *but, when maximum acceleration mode:
               *end of test for acceleration mode.
            *otherwise, it is an arc followed by straight segment:
               *when shorter turn is not in the specified direction:
                     *change turn direction to match specified dir;
               *end of test if shorter turn is not as specified.
               *when position is too close given the turn radius:
                  *when user specified point:
                     *issue warning message;
                     *force path computation with smaller turn radius;
                  *otherwise, not a user specified point:
                     *remove the new point;
                  *end of test for user specified point.
               *end of test for point too close.
               *when computing path to the new point;
                  *compute center of arc, vector from ctr to new pt.;
                  *normalize this vector;
                  *invoke logic to compute path with arc and line;
                  *store the turn radius using + for left turns;
                  *if inserted intermediate point on arc, give it
                  * same turn radius as LPREV;
               *end of test for computing path.
            *end of test for type of path.
         *end of test for allowing an arc.
      *end of test for different points.
   *end of logic for DYNPTH.
```

Subroutine DYNVEC is a utility module which defines the plane of maneuver, i.e. the plane which contains the straight line or circular arcs for the specified path points. This is the top-level design for Subroutine DYNVEC:

```
ABSTRACT     movement utility to compute normal and radial vectors
   *begin logic to normal and radial vectors:
      *compute vector from first point to second and the velocity;
      *normalize this (range) vector;
      *compute the vector normal to plane of maneuver;
      *normalize the vector perpendicular to plane of maneuver;
      *compute radial vector and normalize it;
      *when the path is linear (no turn required):
         *return zero radial and normal vectors;
         *when the second point is not straight ahead:
            *invent a normal vector;
            *compute radial vector;
            *normalize this radial vector;
         *end of test for second point straight ahead or behind.
      *otherwise, a turn is involved:
         *compute magnitude of circular turn radius;
      *end of test for linear or circular movement.
   *end of logic for DYNVEC.
```

When a turn is necessary to reach the specified path point, Subroutine DYNTRN is used to compute and add the appropriate data to the path of the mover. This is the top-level design for Subroutine DYNTRN:

```
ABSTRACT     addition to future path list is arc followed by line
```

```
   *begin logic to add arc and line to movement path:
      *when new position is on the arc:
         *invoke logic to normalize velocity vector;
         *compute the distance around the arc;
         *when using constant (default) acceleration:
            *compute the time to travel around arc;
         *otherwise, using maximum acceleration:
         *end of test for type of acceleration.
      *otherwise, path is arc followed by straight line:
         *allocate FPL entry for end of arc;
         *add to future path list just before the last point;
         *compute length of line from end of arc to new point;
         *compute another vector and normalize it;
         *when turning away from the target;
         *normalize vector from center of arc toward end of arc;
         *store point at end of arc and start of line segment;
         *compute vector from end of arc to new point;
         *normalize the velocity vector and store at end of arc;
         *compute distance around arc, total distance between pts;
         *when using constant (default) acceleration:
            *compute and store time at new point;
            *compute constant acceleration value;
            *compute time to travel accounting for acceleration;
            *store time and speed at end of arc;
         *otherwise, using maximum acceleration:
            *calculate time around the arc;
            *calculate time on linear segment;
         *end of test for type of acceleration.
      *end of test for position on arc or arc followed by line.
   *end of logic for DYNTRN.
```

Subroutine MAXACC is used to compute the time needed to reach a new path point based on the acceleration limits of a mover. This is the top-level design for Subroutine MAXACC:

```
ABSTRACT     find time to next point using maximum acceleration
   *begin logic to find time to next point using maximum acceleration:
      *when speeds are different:
         *account for possible deceleration to next point;
         *calculate time and distance needed to ac(de)celerate;
         *when distance remains after ac(de)celeration completed:
            *when not turning:
               *create intermediate point linearly at end of
               *ac(de)celeration;
            *otherwise, account for arc:
               *create intermediate point on arc at end of
               *ac(de)celeration;
            *end of test for turning.
            *when new speed not zero:
               *link intermediate and new points;
               *time between points equals ac(de)celeration time plus
               *constant speed time;
            *otherwise:
               *copy intermediate point data into new point;
            *end of test for new speed of zero.
         *otherwise, distance not long enough to reach desired speed:
            *calculate time needed to travel to next point at max.
            *ac(de)celeration;
            *adjust speed at new point to reflect highest(lowest)
                possible;
            *when appropriate, write advisory indicating unreachable
                speed;
         *end of test for remaining distance after ac(de)celeration.
```

```
     *end of test for different speeds.
  *end of logic for MAXACC.
```

2.2.4 Assumptions and Limitations

- The paths and orientations of moving players are assumed to be adequately modeled using a 3-DOF representation with orientation aligned with the velocity vector.
- The movement algorithms assume that a single number is adequate to represent the acceleration limits of an aircraft. In reality, acceleration limits can vary due to such factors as weather conditions and total load. The single value may therefore be unrealistic for some flight conditions.
- Movement paths are represented by a series of straight line segments and arcs of circles.
- There is no explicit capability for multiple players to move in formation.

2.2.5 Known Problems or Anomalies

Movement arcs for turns are computed from the current player location and direction to the desired location using the shortest arc. This calculation disregards terrain obstructions and may result in inadvertent terrain collisions when a player is flying at low altitudes.

The TIME-WINDOW modifier for a planned path can cause path points to be computed with negative speed values. This can cause an infinite loop in the scheduling of sense events for sensors trying to detect this moving player.